Sissejuhatus infotehnoloogiasse

Ülevaade loengust

Keeled: põhiideed, näited

Cache ja mäluhierarhia

Operatsioonisüsteemid

- Põhieesmärgid
- Põhitehnoloogiad
- Tüübid
- Ülevaade peamistest OS-idest

Ecki assembler

This program counts. It starts by putting the number 1 into memory location 12, and then it adds one to the number in that location over and over, forever.

```
LOD-C 1
STO 12
LOD 12
INC
STO 12
JMP 2
```

```
        LOD-C 1     ; Set Count equal to 1
        STO Count
Loop:   LOD Count   ; Add 1 to Count
        INC
        STO Count
        JMP Loop    ; Jump back to start of loop

        @12
Count:  data ;  Location to be used for counting
```

Alamprogrammide kasutamine

```
        lod-c 13  ; Set up to call the subroutine with sto N1
                  ; N1 = 13, N2 = 56, and ret_addr = back.
        lod-c 56
        sto N2
        lod-c back
        sto ret_addr
        jmp Multiply     ; Call the subroutine.

back:   lod Answer   ; When the subroutine ends, it returns
                     ;    control to this location, and the
                     ;    product of N1 and N2 is in Answer.
                     ;    This LOD instruction puts the answer
                     ;    in the accumulator.

        hlt    ; Terminate the program by halting the computer
```

Hierarhia pistikutest assemblerini

Esimene programmeerimismeetod: kaablid ja pistikud

Teine: von Neumanni arhitektuur, programm mälus binaarkoodina:

```
010111010100101 101010101001
110101011010100 101010010100
111010100101001 110101111010
101010100101001 110011010101
110101001010010 101001000111
101001011101010 110101001001
```

(arvude liitmine 0…n)

Hierarhia...

Variant eelmisest, kus vaja vähem kirjutada:

```
AB05 E3D5
CD01 032A
4BD0 CDE1
```

siin kasutasime nelja biti kodeeringut (hex, sest 16 varianti)

0000 = 0
0001 = 1
0010 = 2
0011 = 3
0100 = 4
0101 = 5
0110 = 6
0111 = 7
1000 = 8
1001 = 9
1010 = A
1011 = B
1100 = C
1101 = D
1110 = E

Sumto MIPS-I (SGI spinoff) assembleris

Argumendid registritesse $4 ja $8
Resultaat registrisse $2

```
sumto:                      ; Register $4 on n
        li   $3, 0          ; Register $3 on summa
        li   $2, 0          ; Register $2 on i
        blt  $4, $0, L3     ; Kui n<0 mine L3
L5:     addu $3, $3, $2     ; sum = sum + i
        addu $2, $2, 1      ; i = i + 1
        ble  $2, $4, L5     ; Kui i<=n mine L5
L3:     move $2, $3         ; Sum sisaldab resultaati.
        Jr   $31            ; Mine aadressile registris
   $31
```

Sumto ja Sun Sparc'i assembler

Sparc saadab argumendid registrites %o0 kuni %o7 ja resultaadi %o0

Instruktsioon peale hüpet tehakse alati

```
_sumto:                     ; Register %o0 on n.
        mov %o0,%g3         ; Salvesta n registrisse %g3.
        mov 0,  %o0         ; Register %o0 on nyyd sum.
        cmp %o0,%g3         ; Kui 0>n ...
        bg  L3              ; ... mine L3
        mov 0,  %g2         ; ,aga enne i=0.
        add %o0,%g2,%o0     ; sum = sum + i.
L5:     add %g2,1  ,%g2     ; i = i + 1.
        cmp %g2,%g3         ; Kui i<=n ...
        ble,a L5            ; ... mine L5
        add %o0,%g2,%o0     ; ,aga enne sum = sum + i.
L3:     retl                ; Valmis...
nop                         ; ,aga enne ära tee midagi!
```

Sumto ja Intel 386, 486, Pentium, ...

386 on vähe registreid, argument saadetakse hariliku mälu kaudu. Resultaat saadetakse registris %edx.

```
_sumto:
        pushl %ebp              ; Loome ''framepointer''-i
        movl  %esp,%ebp         ;
        movl  8(%ebp),%ecx      ; Võta n.
        xorl  %eax,%eax         ; sum = 0
        xorl  %edx,%edx         ; i = 0
        cmpl  %ecx,%eax         ; Kui i>n ...
        jg L3                   ; ... mine L3
        .align 2
L5:     addl  %edx,%eax         ; sum = sum + i
        incl  %edx              ; i = i+1
        cmpl  %ecx,%edx         ; Kui i<=n ...
        jle L5                  ; ... mine L5
L3:     leave                   ; Taasta ebp.
        ret                     ; Valmis!
```

Kõrgkeeled

Automatiseerivad ja lihtustavad hulga “harilikke” protseduure, mida assembleris programmeerides vaja

Ei anna assembleriga analoogilist kontrolli masina üle

Kõrgkeeled on erineva abstraktsusastmega:

Masinalähedane ja ebamugav: Fortran, C (portaabel assembler)

Abstraktsem ja mugavam: Lisp, Ada, ML, Java, Python, ...

Kuidas keeles X kirjutatud programmi täidetakse?

Pea silmas, et: arvuti suudab täita ainult masinkoodis programme.

On olemas kaks põhivarianti keeles X programmi täitmiseks.

Kompileerimine: masinkoodis programm nimega kompilaator teisendab keeles X programmi masinkoodfailiks Y. Seejärel täidetakse saadud masinkoodis programm Y. Näide: C.

Interpreteerimine: masinkoodis programm nimega interpretaator loeb sisse X keeles faili ja asub seda rida-realt täitma. Näide: vana BASIC.

NB!

Programmi interpreteerimine on ca 10-200 korda aeglasem, kui kompileeritud koodi täitmine.

Põhimõtteliselt saaks igas keeles kirjutatud programme nii interpreteeritult täita kui kompileerida.

Praktikas eelistatakse vahel interpreteerimist, vahel kompileerimist.

Kuidas keeles X kirjutatud programmi täidetakse?

Kompromissvariante:

Kompilaator kompileerib X faili **vahekoodiks** Y, seejärel interpreteeritakse vahekoodi Y (Python, Java).

Interpretaator interpreteerib vahekoodi Y, kuid **kompileerib töö ajal osa Y-st masinkoodiks**, mida seejärel täidab (Java) nn just-in-time compilation ehk **JIT**.

Kompileeritava programmi valmimine

Olgu meil (näiteks C keeles) failid main.c ja swap.c

Teeme gcc main.c swap.c -o minuprogramm

Kompilaator (näiteks gcc) teeb järjest mitut eri asja:

- **Kompileerimine**
  - Kompilaator teeb neist assemblerikeelsed ajutised failid
  - Kompilaator teeb assemblerfailidest masinkood+sümbolinfo failid
- **Linkimine**
  - Linkur otsib kokku vajalikud olemasolevad failid osa sümbolinfo seostamiseks päris koodi-viidetega

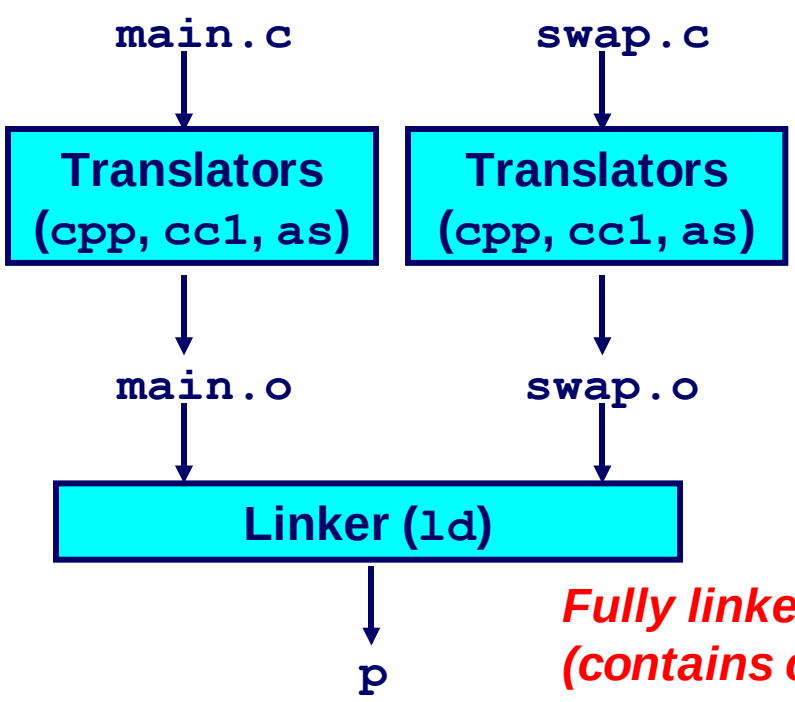
Käivitame saadud programmifaili minuprogramm:

- Opsüsteemi **loader** otsib lisaks vajalikud olemasolevad failid osa sümbolinfo seostamiseks päris koodi-viidetega
- Saadud kogum paigutatakse mällu, tehakse opsüsteemi infoblokk tema jaoks (protsess) ja kogum käivitatakse

(CMU Bryant & Hallaron course) Static Linking

Programs are translated and linked using a *compiler driver*:

```
unix> gcc -O2 -g -o p main.c swap.c
unix> ./p
```

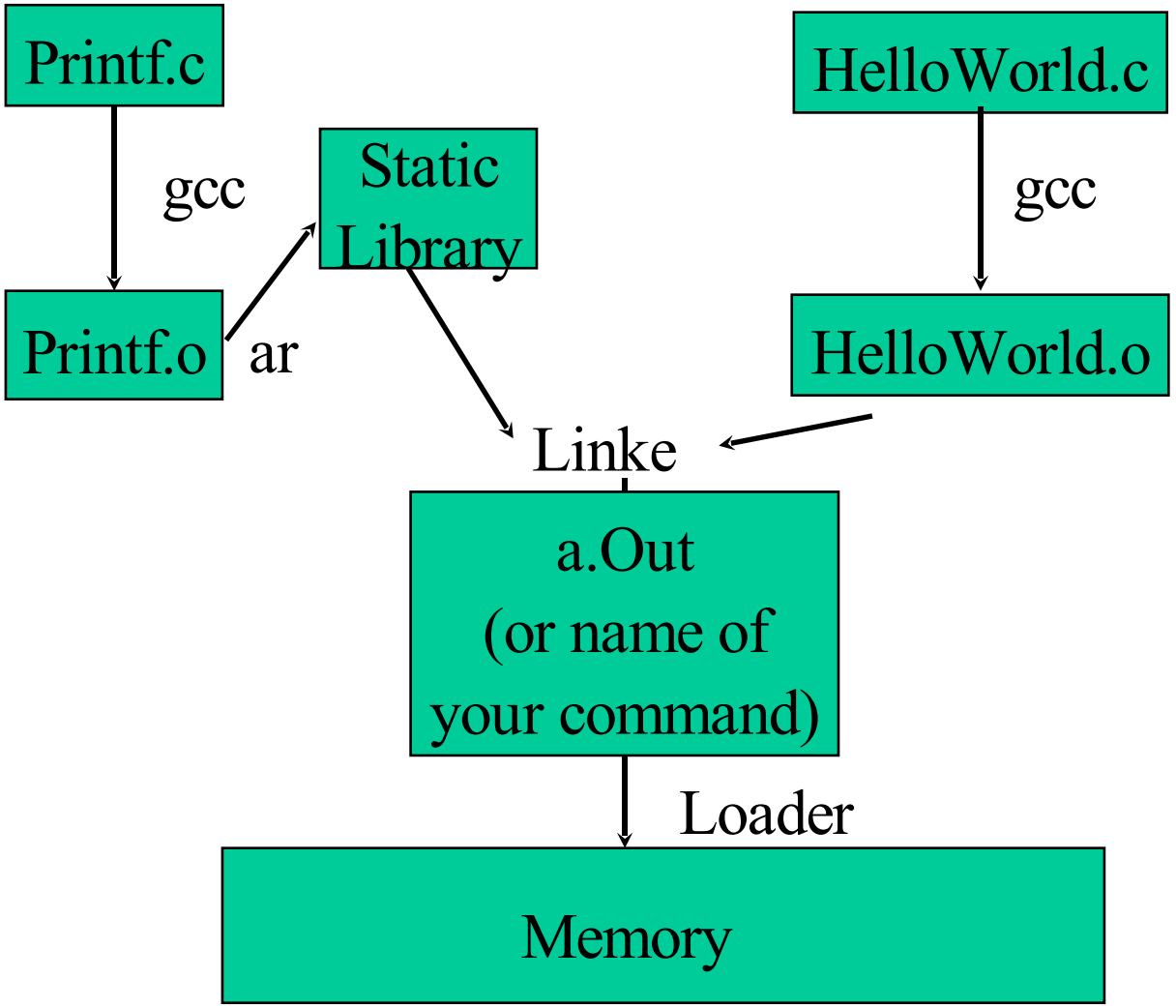


Source files

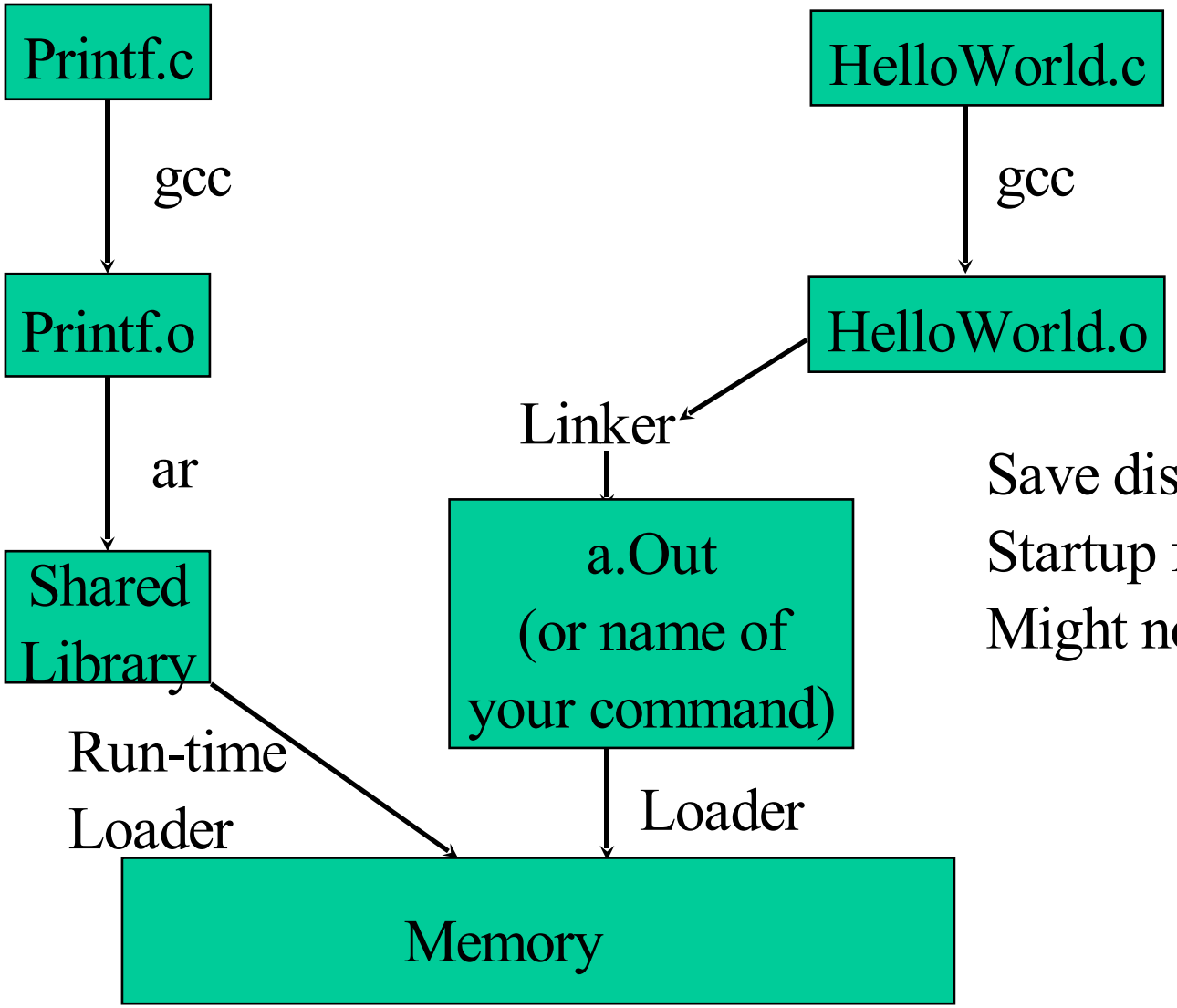
Separately compiled relocatable object files

Fully linked executable object file (contains code and data for all functions defined in main.c and swap.c

(CMU B&H ..) Static Linking and Loading



(CMU B&H ..) Run-time Linking/Loading



Save disk space.
Startup faster.
Might not need all.

Keelte erisused: kolm põhiasja

Süntaks (kuidas kirjutatakse näiteks **if .. then .. else** ühes või teises keeles)

Semantika ehk tähendus (mida õigesti kirjutatud programm tegelikult siis teeb)

Teegid (libraries) (millised **valmisprogrammijupid** on selle keele jaoks kergesti kättesaadavad või kohe kaasa pandud)

Keeled: tüüpilised asjad, mida pea iga keel pakub

Primitiivsed andmetüübid:

int, char etc (näiteks: 1 ja –3 on int-id, ‘c’ ja ‘a’ on char-id)

string (näiteks “aaa123bb”)

massiiv (näiteks a[1]=2; a[2]=20; a[3]=15; y=2; x=a[y]+a[1]+3;)

Avaldised:

näiteks x = (y*2) – (5+x);

Elementaarsed juhtkonstruktsioonid:

valik: if ... then ... else

tsükkel: while(x<10) x=x+1;

Funktsioonid:

defineerime: int kuup(int x) { return x * x * x}

kasutame: x = kuup(1+kuup(3))+kuup(y);

kasutame rekursiivselt:

```
int fact(int x) { if (x<=0) return 1; else return x*(fact(x-1)); }
```

Keeled: näited lisavõimalustest eri keeltes

Kiired bitioperatsioonid, otsepöördumine mälu kallale: C

Keerulisemad andmetüübid: listid, hash tabelid jne: Lisp, Scheme, Python

Erikonstruktsioonid stringitöötluseks: Perl, PHP

Objektid: C++, Java, C#, Python, Lisp

Moodulid (enamasti ühendatud objektidega): C++, Java, C#

Veatöötluse konstruktsioonid (exceptions): Python, Java, C#

Prahikoristus: kasutamata andmed visatakse välja (Java, Python, Lisp, ...)

Sisse-ehitatud tugi paralleelprogrammide jaoks: Java, C#

Reaalaja-erivahendid: Ada

“Templates” (programm tulemuse sees): PHP, JSP, Pyml

Uute programmide konstrueerimine töö käigus: Lisp, Scheme

Loogikareeglid: Prolog

“laisk” viis funktsioone arvutada: Miranda, Hope, Haskell

Pattern matching (viis funktsioone defineerida): ML, Haskell

jne...

Fortran: summeeri arve 0...n

FORTRAN

```
      INTEGER FUNCTI0N sumto(n)
      isum = 0
      DO i 10 = 0,n
      isum = isum + i
10    CONTINUE
      sumto = isum
      RETURN
      END
```

LISP: summeeri arve 0...n

LISP

```
(defun sumto (n)
   (if (= 0 n)
       0
       (+ n (sumto (- n 1))) ))
```

Sumto ja Modula-2

Modula-2

```
PROCEDURE sumto(n:INTEGER):INTEGER;
VAR sum,i:INTEGER;
BEGIN
      sum:=0;
      FOR i:=0 TO n DO
        sum:=sum+i
      END;
      RETURN sum
END sumto;
```

Sumto ja Ada

Ada

```
function sumto(n: in INTEGER) return INTEGER is
    sum : INTEGER := 0;
begin
    for i in 0..n loop
       sum := sum + i;
    end loop;
return sum;
```

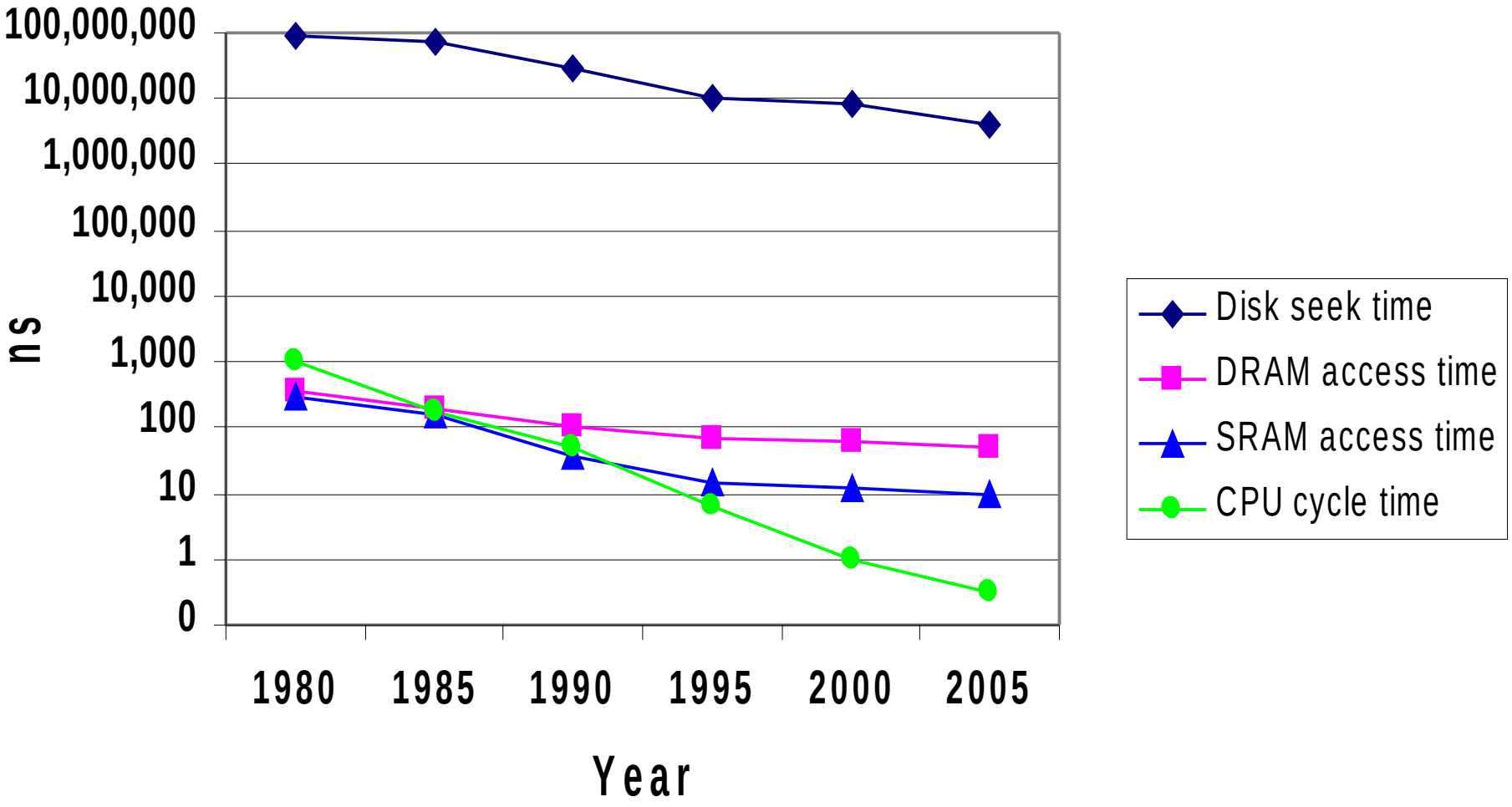
Sumto ja C

C (ja Java)

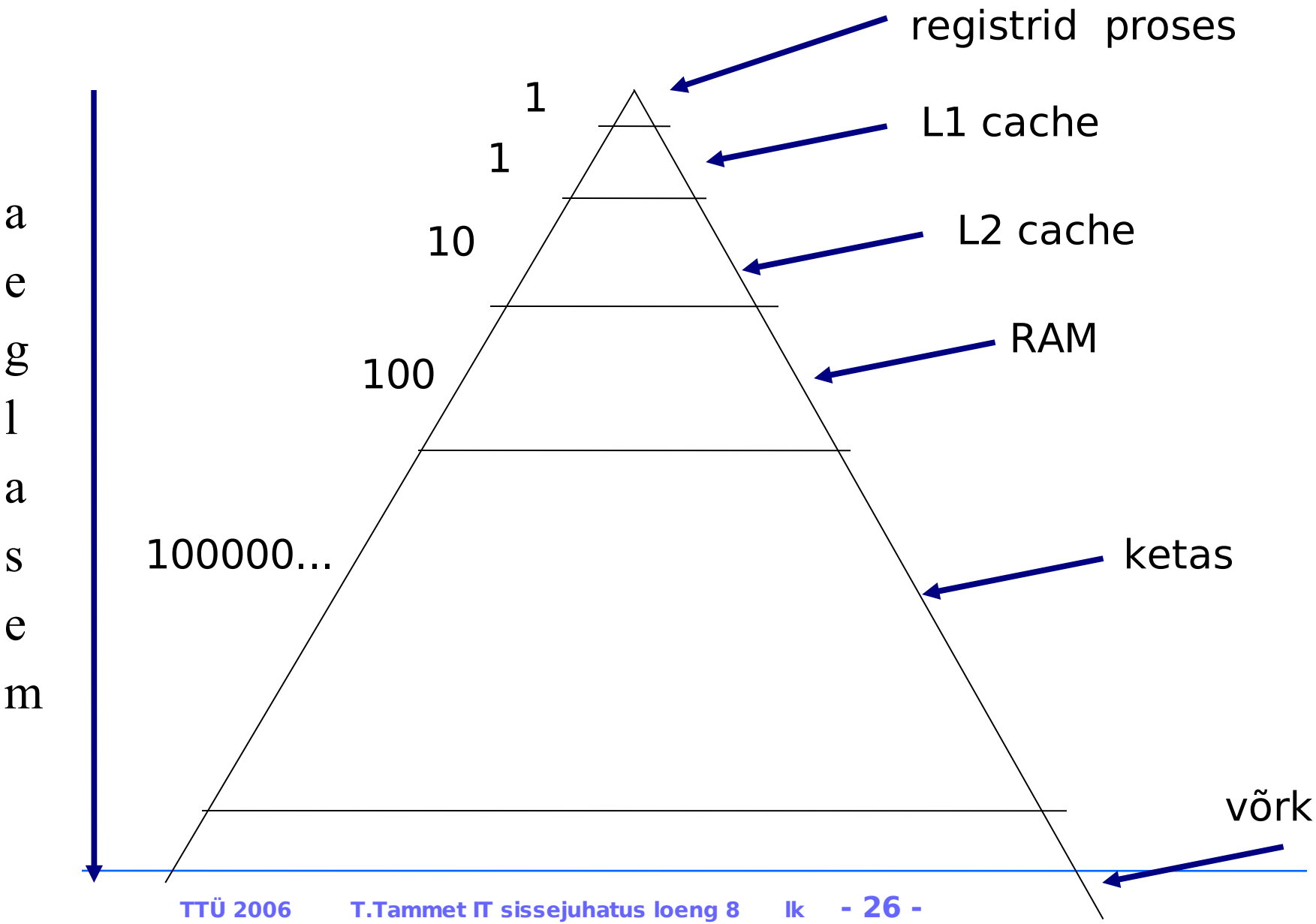
```
int sumto(int n) {
  int i,sum = 0;
  for(i=0; i<=n; i=i+1)
    sum = sum + i;
  return sum;
}
```

(CMU B&H ..) The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.



Cache ja mälu hierarhia



Miks opsüsteem?

Opsüsteemi põhieesmärgid:

Pakkuda programmeerijale valmistehtud standardtükke.

Võimaldada kasutajal arvutis ühtemoodi ja harjumuspäraselt tegutseda, sõltumatult sellest, mis programmid tal arvutis on.

Miks opsüsteem?

Arvutit saaks programmeerida ka ilma opsüsteemita. Sel juhul:

oleks iga programmi tegemine palju raskem kui opsüsteemi olemasolu korral.

kasutajate jaoks näeks eri programmid väga eri moodi välja.

Mida opsüsteem enamasti teeb?

Oskab kettalt programme lugeda ja neid käima panna.

Oskab programme seisma panna (lõplikult või ainult väikese pausi jaoks)

Oskab kettale faile ja katalooge kirjutada ja sealt neid lugeda.

Oskab välisseadmetega (printer, monitor, klaviatuur jne jne) suhelda.

Oskab võrguga suhelda.

jne

Kui opsüsteemi ei oleks, peaks iga programm kõiki neid asju ise teha oskama!

Naked machine

What is there in the naked machine (unclothed by an OS)?

CPU (registers, opcodes),
Memory (may be several chunks),
Devices (able to transfer data to and from memory).

Definitions of OS!? ...

Extended machine, or virtual machine. Takes the basic hardware, and provides additional layers of functionality. The virtual machine is usually simpler and more consistent than the real machine. Practically, additional layers will build on the lowest-level ones, providing a rich programming model for the applications software (which in turn provides a 'virtual machine' to the user).

Resource manager. Allocates scarce resources, like disk space, memory, processor between competing processes and users. Each process thinks that it has its own machine (its own files, its own printer, whatever). In a way, this is also extending the machine (in order to appear to be many machines).

Anything which runs in the computer's privileged mode, or supervisor mode. Typically most modern processors have two or more operating modes, user and supervisor. In supervisor mode, able to access all of memory and all hardware.

Definitions of OS!?

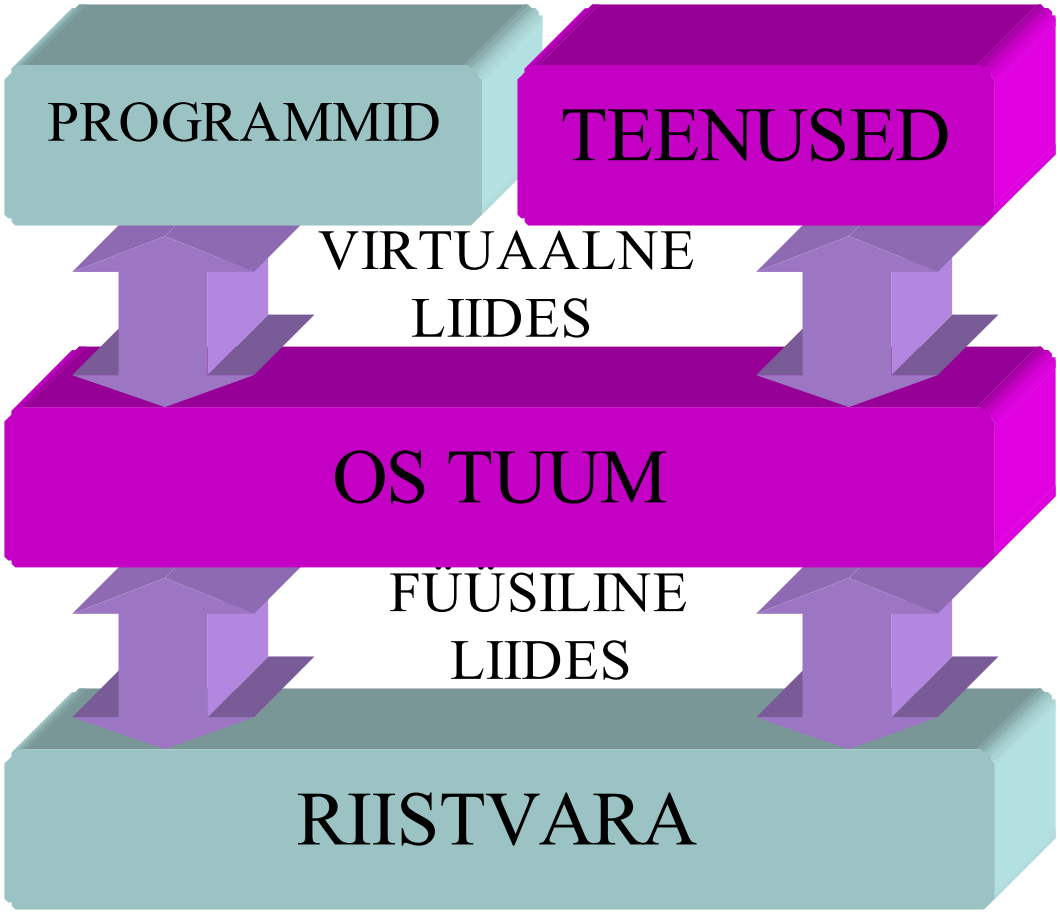
The software which is supplied with the machine. Often includes many things not traditionally considered part of the operating system, like GUIs, editors, Web browsers, and stuff.

A platform on which other software can run. Lots of platforms are not operating systems (like Netscape Navigator, or Microsoft Word). Java is almost an operating system. Provides many services commonly associated with Operating Systems, like managing multiple concurrent processes, and providing access to files.

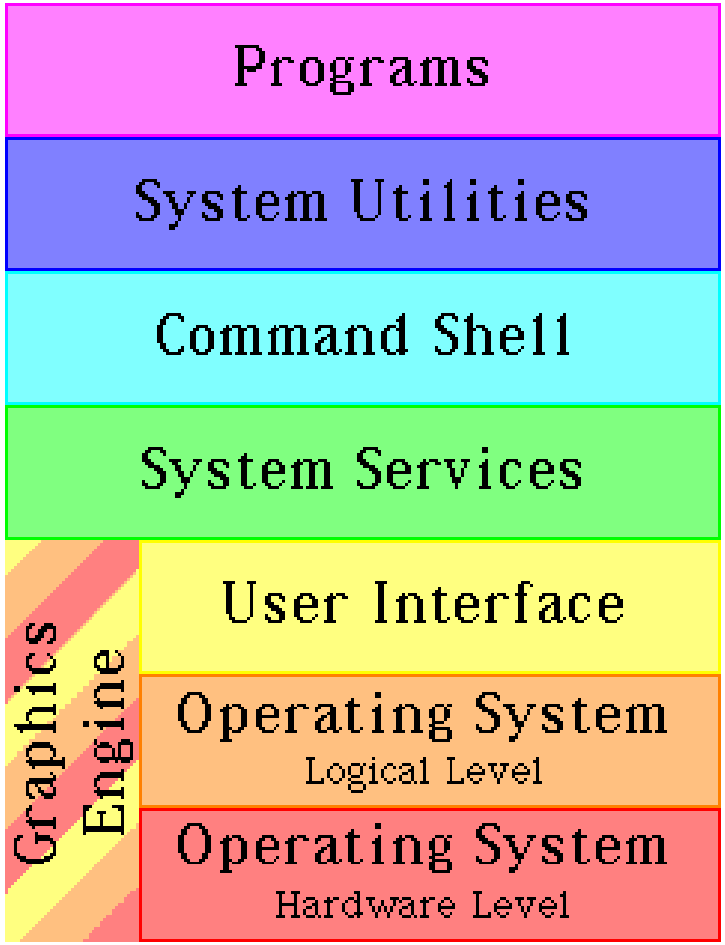
Operatsioonisüsteemide arengulugu

Arvutid ilma OS-ita
Batch processing süsteemid
Multiprotsessing ja terminalid - UNIX
Personaalarvutid - IBM, Xerox, Apple, DOS, Windows

Operatsioonisüsteemi roll



OS layers


Programs
System Utilities
Command Shell
System Services
Graphics Engine
User Interface
Operating System
Logical Level
Operating System
Hardware Level

Operatsioonisüsteemide omadused

Funktsionaalsus

- Protsesside haldamine
- Mälu haldamine
- Failisüsteemid
- Võrguprotokollid
- Kasutajate haldamine

Interrupts

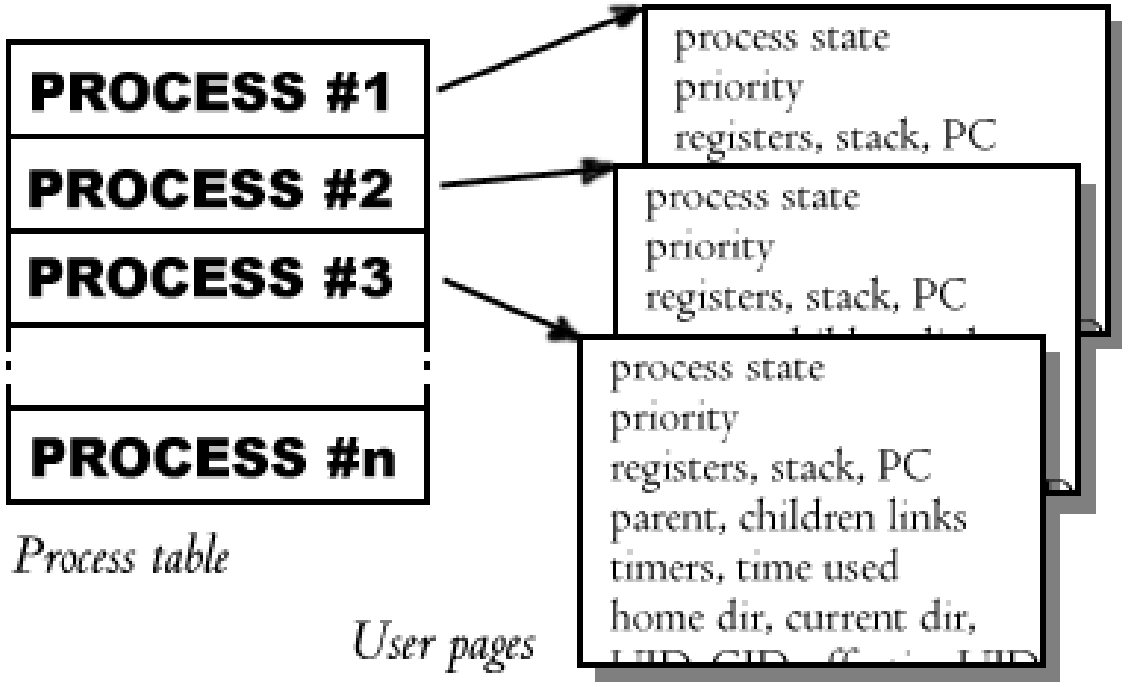
The interrupt handler must save the machine state, do some processing, then call the process scheduler and dispatcher.

When an interrupt occurs

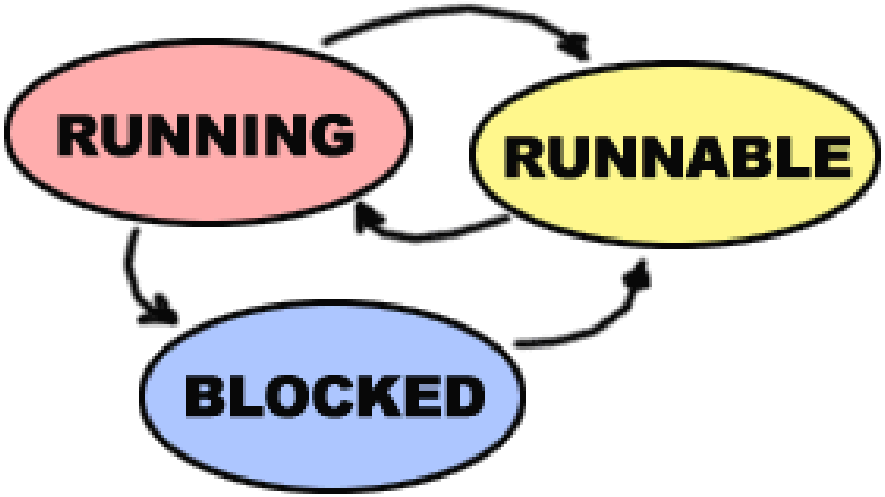
1. the processor hardware makes a quick copy of the program counter and CPU registers
2. the hardware switches to kernel mode and jumps to the interrupt service routine
3. the ISR is usually very short. It may inform a device driver that it received the interrupt; it may just increment some clock counters.
4. next the ISR calls the scheduler, which decides which process to run
5. the scheduler calls the dispatcher, and new process (or maybe the same process) resumes where it left off

An important goal of the OS is to hide interrupts from the user---and from user-level processes.

Processes have their own memory areas



Processes wait and run – when chance comes



Oveview of main OS-es: main Microsoft OS-es

MS-DOS is a text-based desktop operating system made by Microsoft that runs on Intel 80x86. Only low end servers can run on this operating system.

Windows 98, Windows 95, and Windows 3.1 are desktop operating systems made by Microsoft that run on Intel Pentium and Intel 80x86. Only low end servers can run on these operating systems.

Windows NT, Windows NT Server, and Windows NT Server Enterprise Edition are server and workstation operating systems made by Microsoft that run on Intel Pentium, Intel 80x86, and DEC Alpha.

Windows 2000 Professional, Windows 2000 Server, and Windows 2000 Advanced Server are server and workstation operating systems made by Microsoft that run on the Intel Pentium.

Windows XP is a server and workstation operating system made by Microsoft that run on the Intel Pentium. Converges 95 and NT/2000.

OS-es: IBM PC os-s

PC-DOS-2000 is a text-based desktop operating system made by IBM as an update of the older MS-DOS. It runs on Intel 80x86. Only low end servers can run on this operating system.

OS/2 is a high performance desktop and high end operating system made by IBM that runs on Intel Pentium and Intel 80x86.

OS-es: Apple

Macintosh OS 9, OS 8, OS 7, and OS 6 are desktop operating systems made by Apple Computers that run on Motorola/IBM PowerPC and Motorola 680x0.

Rhapsody is a UNIX-based operating system that includes capabilities from the NeXT and Macintosh operating systems. Rhapsody was a foundation for OS X of the Macintosh.

Macintosh OS X (ten) is a desktop operating system based on Rhapsody. Macintosh OS X is made by Apple Computers and runs on Motorola/IBM PowerPC.

OS-es: main free UNIXes

LINUX is a free version of UNIX that runs on Intel Pentium, Intel 80x86, Motorola/IBM PowerPC, Motorola 680x0, Sun SPARC, SGI MIPS, DEC Alpha, HP PA-RISC, DEC VAX, ARM, API 1000+, and CL-PS7110.

FreeBSD is a free version of UNIX that runs on Intel Pentium and Intel 80486.

NetBSD is a free version of UNIX that runs on Intel Pentium, Intel 80486, Intel 80386, Motorola/IBM PowerPC, Motorola 680x0, Sun SPARC, HP PA-RISC, DEC VAX, and ARM.

OpenBSD is a free version of UNIX that runs on Intel Pentium, Intel 80486, Intel 80386, Motorola/IBM PowerPC, Motorola 680x0, Sun SPARC, HP PA-RISC, DEC VAX, and ARM.

GNU Hurd is a free UNIX-based operating system.

OS-es: main common commercial UNIXes

Solaris is a UNIX-based operating system made by Sun Computers that runs on Sun SPARC and Intel Pentium.

Sun-OS is an older text-based UNIX that runs on Sun SPARC. Solaris is an enhancement of Sun-OS that includes a graphic user interface.

AIX is IBM's version of UNIX.

HP-UX is a UNIX-based operating system made by Hewlett-Packard that runs on HP PA RISC.

ULTRIX is a UNIX-based operating system made by DEC. ULTRIX has been replaced by Digital UNIX.

Major handheld operating systems

Palm OS
Windows CE
Symbian

OS-es: less common

IRIX is a UNIX-based operating system made by SGI that runs on SGI MIPS.

NeXT is a UNIX-based operating system made by NeXT that runs on Intel Pentium, Intel 80486, and Motorola 68040. OpenSTEP runs on Intel Pentium, Intel 80486, Motorola 68040, Sun SPARC, and HP PA RISC.

MVS is a mainframe operating system made by IBM

VMS is a high performance operating system made by DEC that runs on DEC VAX. OpenVMS is an updated version of VMS.

NetWare “is a dedicated network operating system” that runs on Intel Pentium, Intel 80486, and Intel 80386.

BeOS is a high performance desktop operating system made by Be that runs on Motorola/IBM PowerPC and Intel Pentium. Only low end servers can run on this operating system.

AmigaOS is an old but popular operating system that is being resurrected. Only low end servers can run on this operating system.

How Unix spread

After AT&T was forced to abandon commercial computing as part of an antitrust settlement, AT&T's UNIX was made available for free to the academic community. Because UNIX had been designed in a way that made it easy to "port" (move) to new hardware, colleges and universities that switched to UNIX were able to run a single operating system on all of their computers, even if their computers came from multiple manufacturers.

Eventually UNIX spread into the business community, and pushed aside almost all proprietary mainframe and minicomputer operating systems. Only IBM's MVS and DEC's OpenVMS survived in common use (MVS because of the sheer number of installations using it and OpenVMS in the banking and financial community because of its high reliability, security, and preservation of data). Even IBM and DEC ended up offering their own versions of UNIX as well as their proprietary operating systems.

UNIXi perekonnad kuni 1993

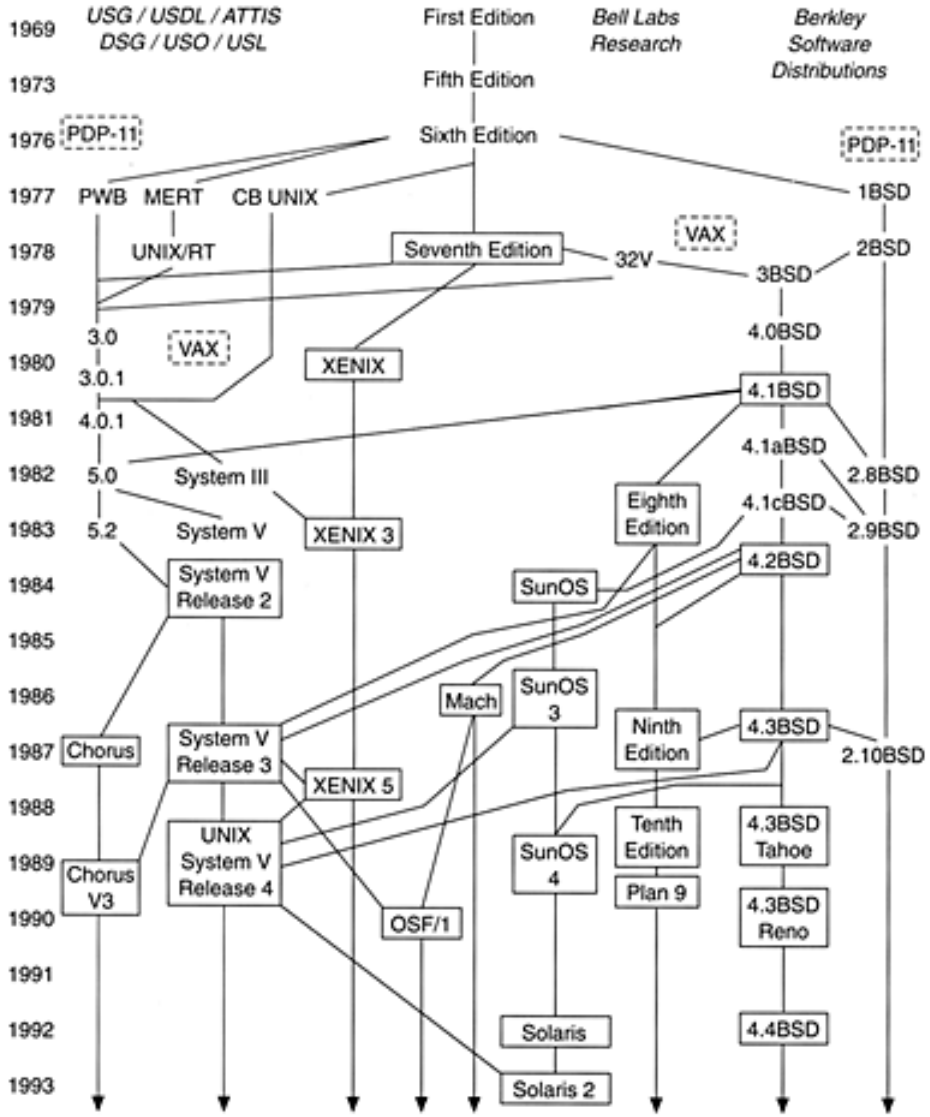


Figure 19.1 History of UNIX versions.

Linux & FreeBSD võrdlus

Linux

- Hajutatud arendus
- Erinevad väljaanded
- Kaasaegsem
- Laiem riistvara toetus
- SysV tüveline
- Kasutajasõbralikum

FreeBSD

- Tuumikmeeskond
- Üks väljaanne korraga
- Stabiilsem
- Väiksem riistvara toetus
- BSD tüveline
- Tehniliselt terviklikum